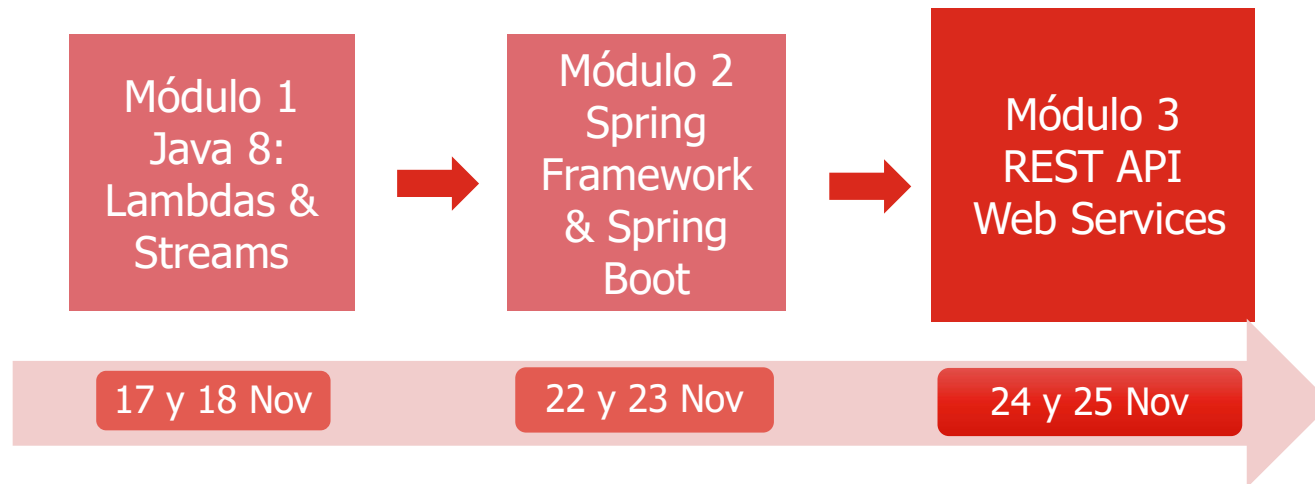


# Programa Juniors Backend GFT



Formador: Ezequiel Llarena Borges

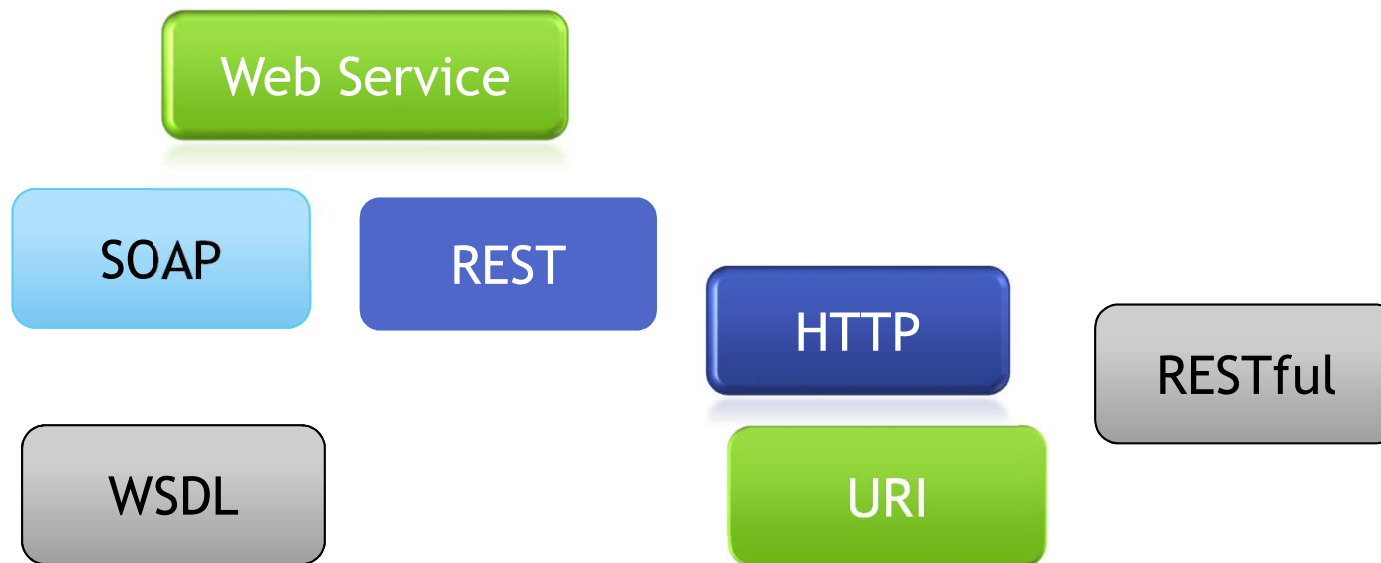


---

La idea es... “ponérselo fácil a los consumidores”

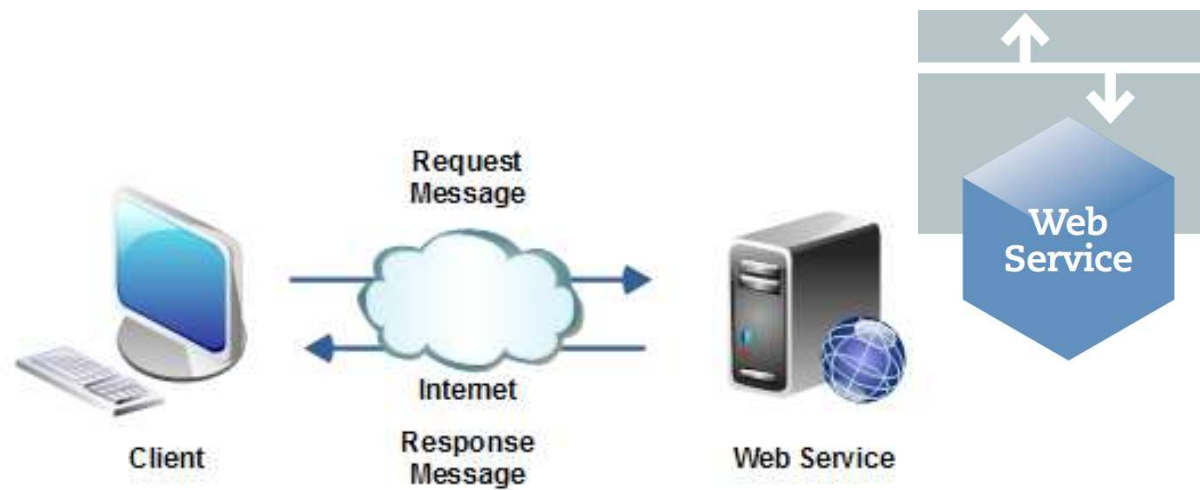
# Desarrollo Servicios REST

Algunos términos...



# Desarrollo Servicios REST

## Servicio Web



# Desarrollo Servicios REST

## Web Service

- **API online** a la que se puede acceder programáticamente
- Código desplegado en diferentes máquinas
- Retorna datos en formato XML o JSON
- Facebook y Twitter publican web services consumidos por otros desarrolladores desde sus códigos
- Apps, Games, ...

# Desarrollo Servicios REST

## Web Service

<http://www.twitter.com>



HTML



<http://api.twitter.com>



XML / JSON

### JSON

```
{
  "siblings": [
    { "firstName": "Anna", "lastName": "Clayton" },
    { "lastName": "Alex", "lastName": "Clayton" }
  ]
}
```

### XML

```
<siblings>
<sibling>
<firstName>Anna</firstName>
<lastName>Clayton</lastName>
</sibling>
<sibling>
<firstName>Alex</firstName>
<lastName>Clayton</lastName>
</sibling>
</siblings>
```

# Desarrollo Servicios REST

## Formato de datos



```
<user>
  <id>1</id>
  <name>Me</name>
  <email>me@gmail.com</email>
</user>
```

content-type: text/xml



```
{
  "id" :1,
  "name" : "Me",
  "email" : "me@gmail.com"
}
```

content-type: application/json

# Desarrollo Servicios REST

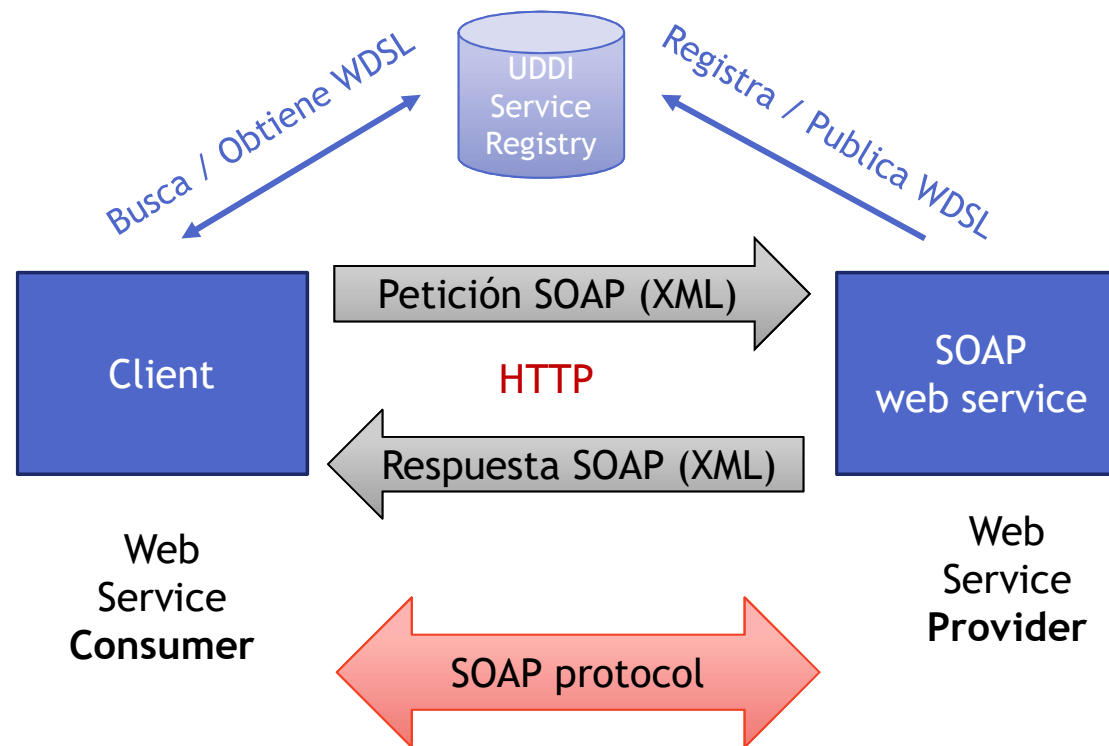
## Estilos de Servicios Web

- **RPC (Remote Procedure Calls)**
  - Llamadas a procedimientos y funciones distribuidas (operación WSDL)
- **SOA (Service-Oriented Architecture)**
  - Comunicación vía mensajes (servicios orientados a mensajes)
- **REST (Representational State Transfer)**
  - Interacción con recursos con estado



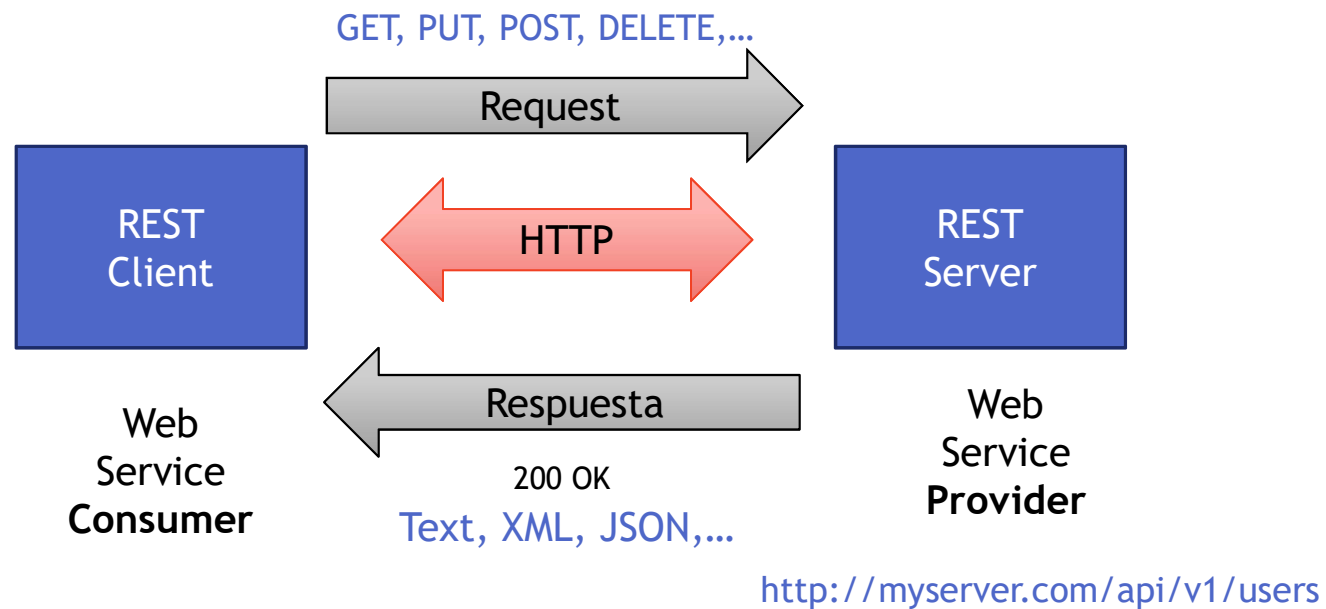
# Desarrollo Servicios REST

## SOAP Web Service



# Desarrollo Servicios REST

## REST Web Service



# Desarrollo Servicios REST

## SOAP vs REST

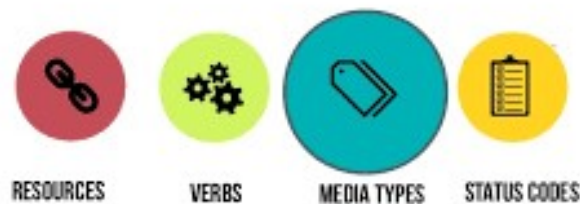
#	SOAP Simple Object Access Protocol	REST Representational State Transfer
1	Protocolo de mensajes XML	Protocolo de estilo arquitectural
2	Usa WSDL en la comunicación entre el consumidor y el proveedor	Usa XML o JSON para enviar y recibir datos
3	Invoca a los servicios mediante llamadas a métodos RPC	Llamada a un servicio vía URL
4	Información que devuelve no legible para el humano	Resultado es legible por el humano (XML, JSON...)
5	Transferencia sobre HTTP y otros protocolos (SMTP, FTP, etc...)	Transferencia es sólo sobre HTTP
6	JavaScript permite invocar SOAP (implementación compleja)	Fácil de invocar desde JavaScript
7	El rendimiento no es tan bueno comparado a REST	Rendimiento mucho mejor que SOAP - menor consumo CPU, código más pulido, etc.

# Desarrollo Servicios REST

## REST (REpresentational State Transfer)

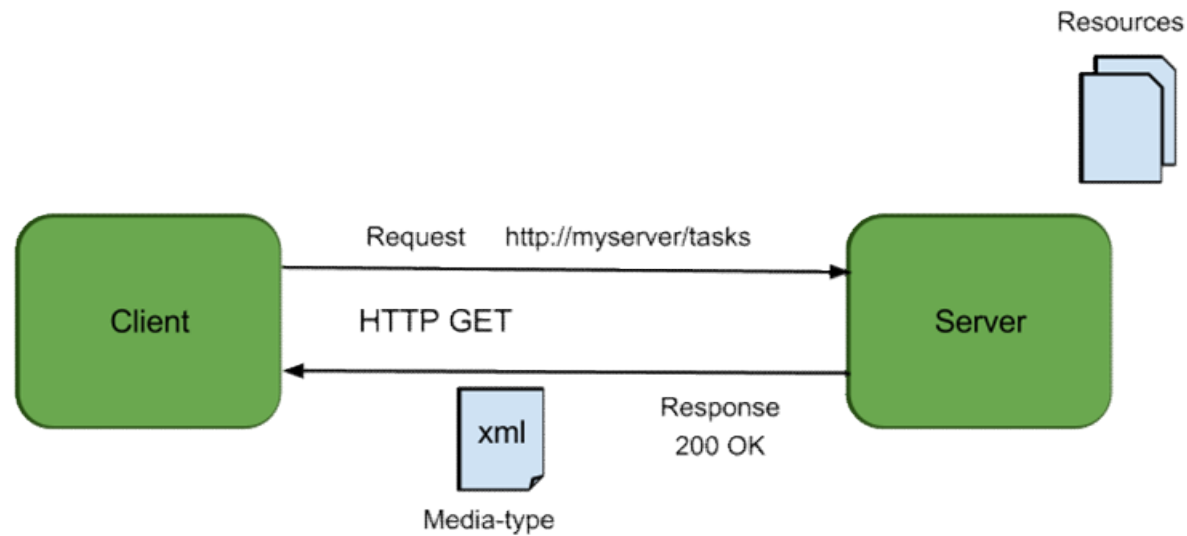
- Estilo de **arquitectura** de software para desarrollar servicios web
- Basado en **estándares web** y protocolo **HTTP**
- REST no es un estándar
- Todo es un **Recurso**
- Un servidor REST permite acceso a los recursos
- Recursos identificados mediante un ID global (**URIs**)
- Diferentes **representaciones** de los recursos (text, XML,JSON)

### REST Style consists of ...



# Desarrollo Servicios REST

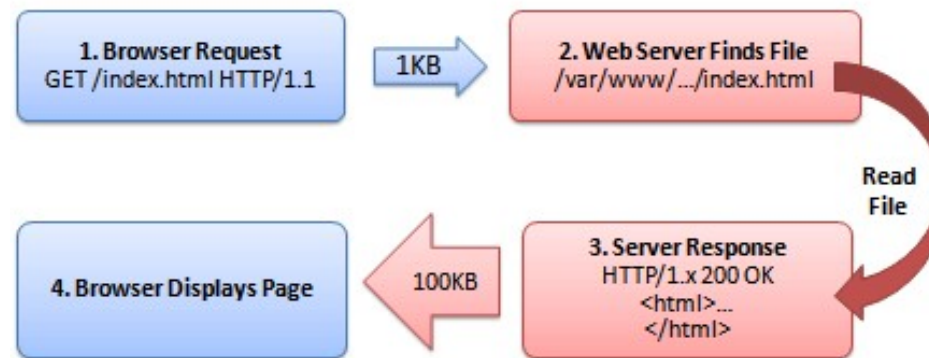
## Funcionamiento de un Web Service REST



# Desarrollo Servicios REST

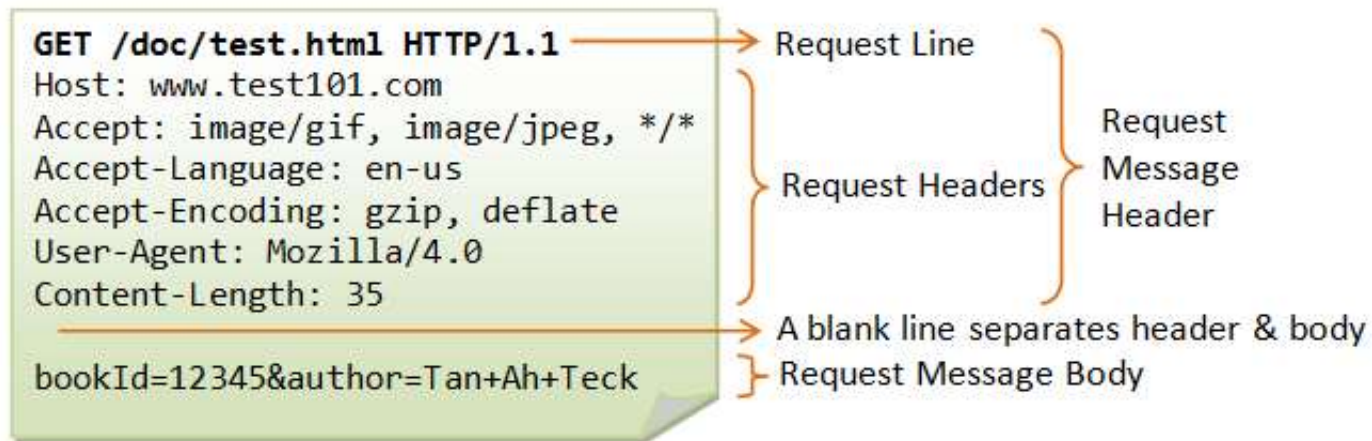
## HTTP

### HTTP Request and Response



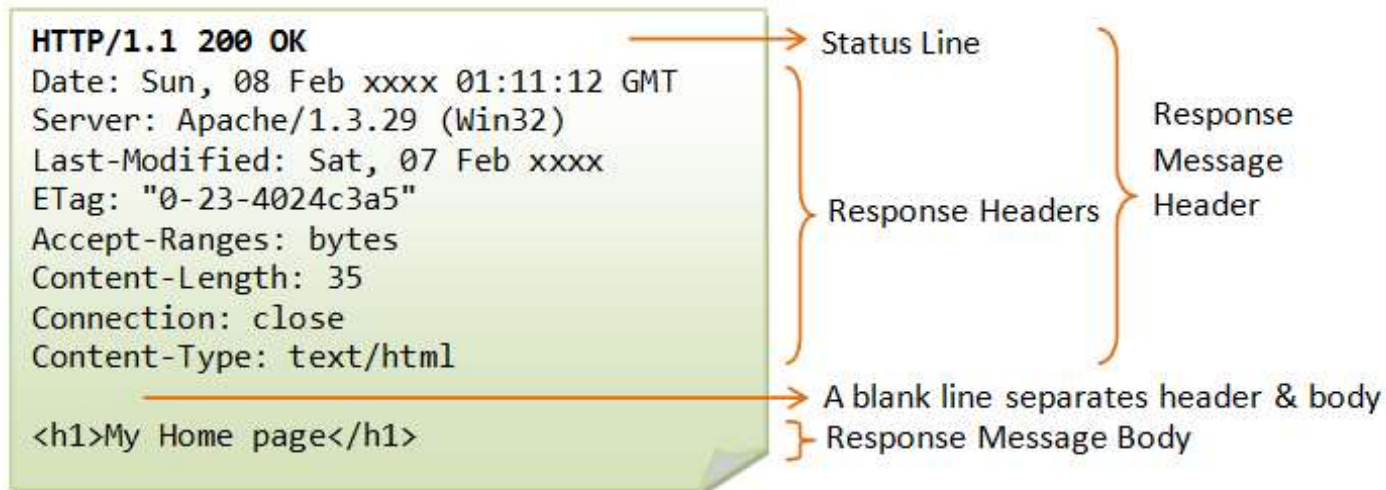
# Desarrollo Servicios REST

## HTTP Request



# Desarrollo Servicios REST

## HTTP Response





# Desarrollo Servicios REST

## REST y HTTP

- Resource based URIs
- HTTP operations
  - ✓ GET
  - ✓ POST
  - ✓ PUT
  - ✓ DELETE
- HTTP status codes (*Metadata*)
  - ✓ 200 - Success (GET)
  - ✓ 201 - Created (POST)
  - ✓ 500 - Server error
  - ✓ 404 - Not found (GET, PUT, DELETE)
- Message headers
  - ✓ Content types: `text/xml`, `application/json`

# Desarrollo Servicios REST

## RESTful

- Define URI base para los servicios
- Basado en métodos HTTP y concepto REST
  - ✓ GET
  - ✓ POST
  - ✓ PUT
  - ✓ DELETE
- Soporta MIME-types
  - ✓ XML
  - ✓ text
  - ✓ JSON
  - ✓ User-defined, ...

# Desarrollo Servicios REST

## Métodos HTTP

GET

- Obtiene un recurso
- Es seguro (no side-effects)
- Resultados “cacheables”
- Idempotente

POST

- Crea un recurso nuevo
- No es idempotente

PUT

- Actualiza un recurso existente
- Idempotente

DELETE

- Elimina un recurso
- Idempotente

# Desarrollo Servicios REST

## Idempotencia

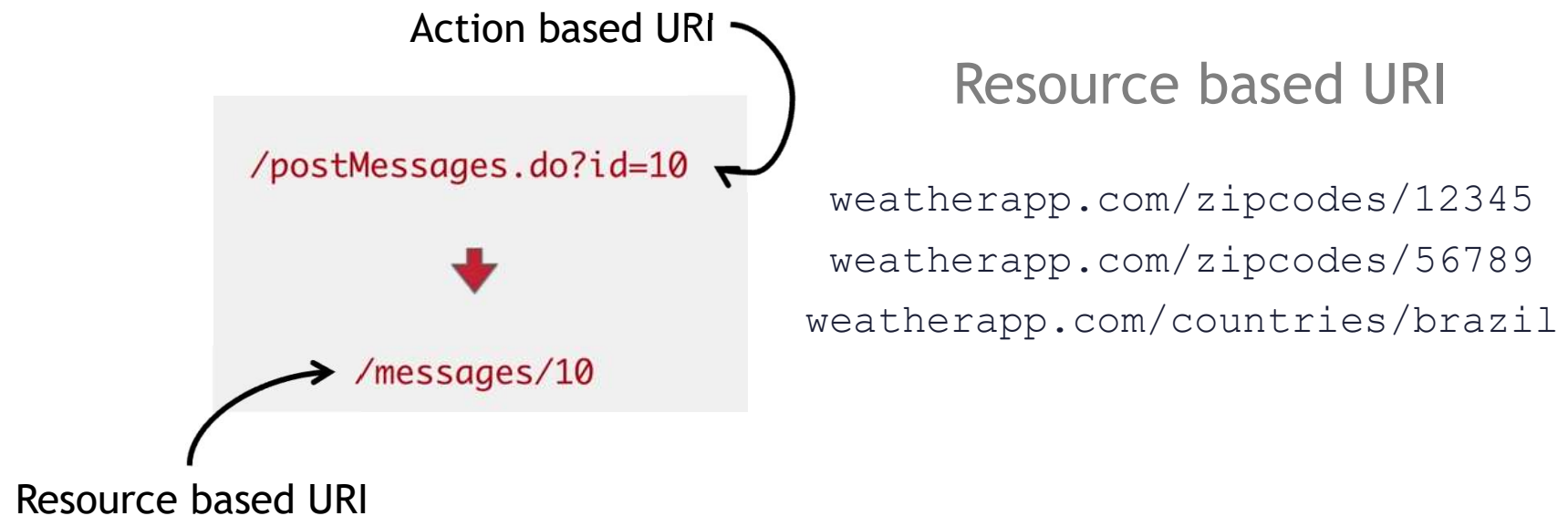
- El **cliente** puede invocar repetidamente un método generándose siempre el mismo resultado.
- Las operaciones idempotentes producen siempre el mismo resultado en el **servidor**.

Idempotente	No Idempotente
<ul style="list-style-type: none"><li>• GET</li><li>• PUT</li><li>• DELETE</li><li>• HEAD</li><li>• OPTIONS</li></ul>	<ul style="list-style-type: none"><li>• POST</li></ul>

# Desarrollo Servicios REST

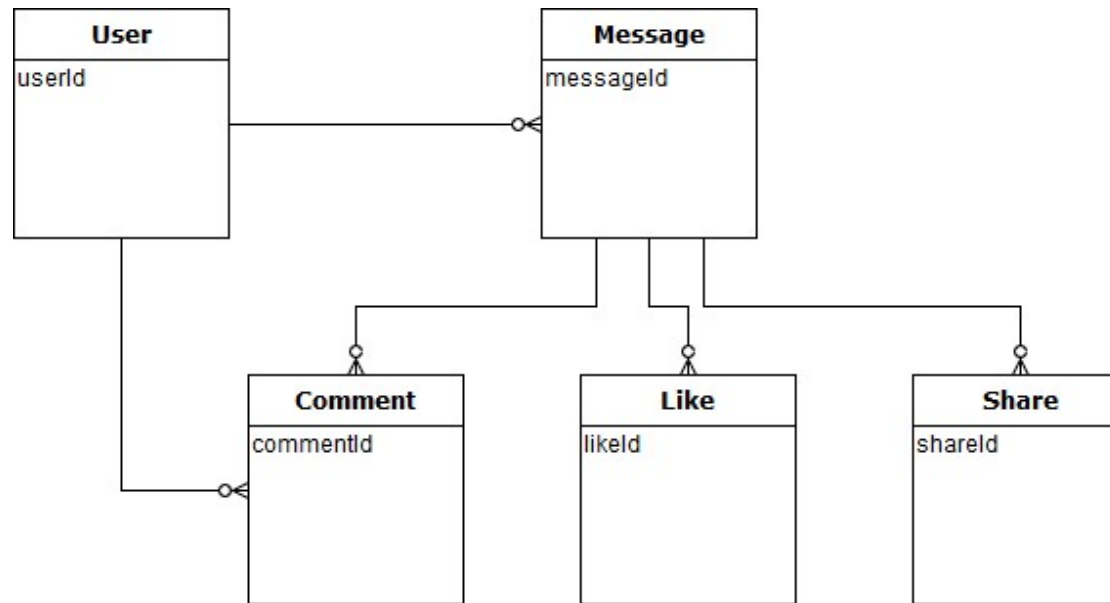
## Resource locations

URI = Uniform Resource Identifier



# Desarrollo Servicios REST

## Diseño de API REST (Ejemplo)



# Desarrollo Servicios REST

## Instance resource URI

instance / item



Resource based URI

/profiles/{profileName}

/messages/{messageId}

/messages/{messageId}/likes/{likeId}

/coordinates/lats/{latVal}

/coordinates/lons/{lonVal}

# Desarrollo Servicios REST

## Collection resource URI

Collection



Resource based URI

/profiles/

/messages/

/messages/{messageId}/likes/

/profiles/{profileName}/comments/



# Desarrollo Servicios REST

## Filtering results

### Query Parameters



`/messages/?offset=5&limit=10`



*starting point*



*page size*

### Custom Filters



`/messages/?year=2017`



*Filtering Collection*

# Desarrollo Servicios REST

## Operaciones HTTP

/getProducts.do?id=10 ❌

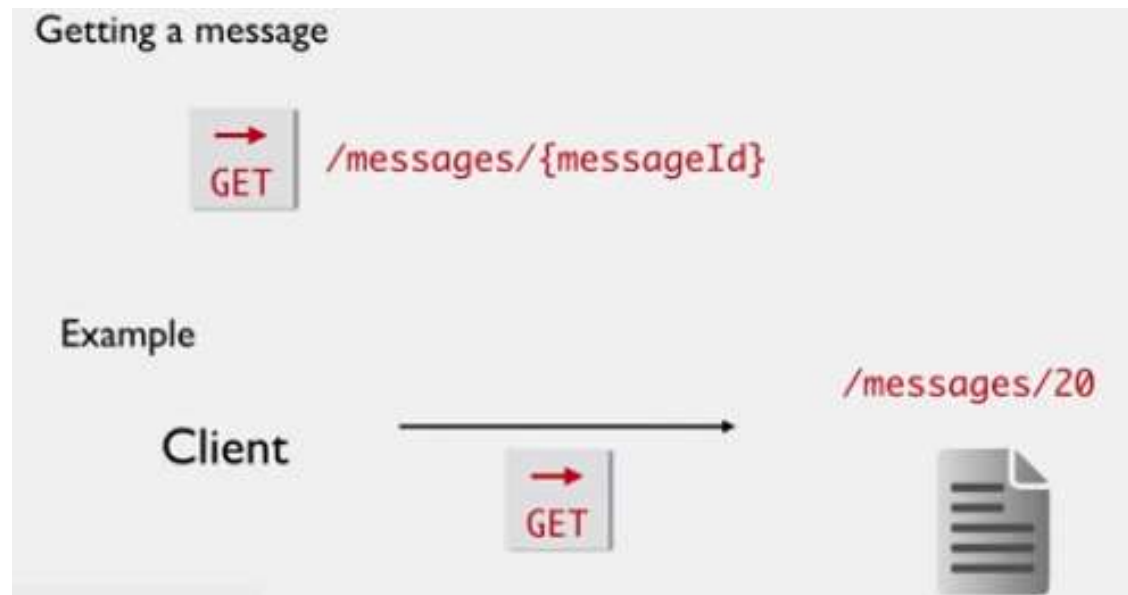
→  
GET /products/10 ✅

/deleteOrder.do?id=10 ❌

→  
DELETE /orders/10 ✅

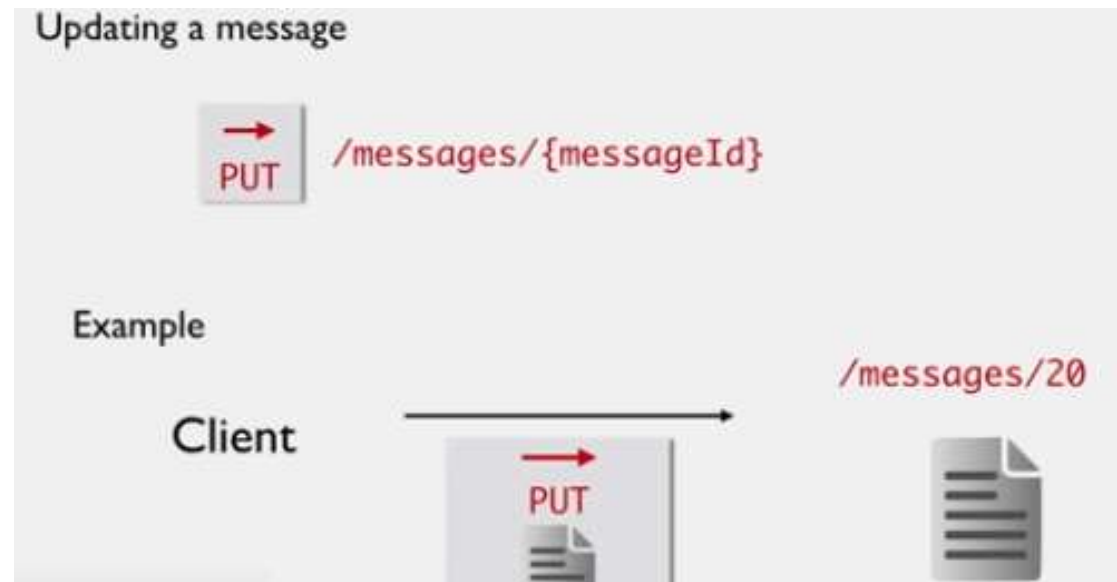
# Desarrollo Servicios REST

## Operaciones HTTP GET



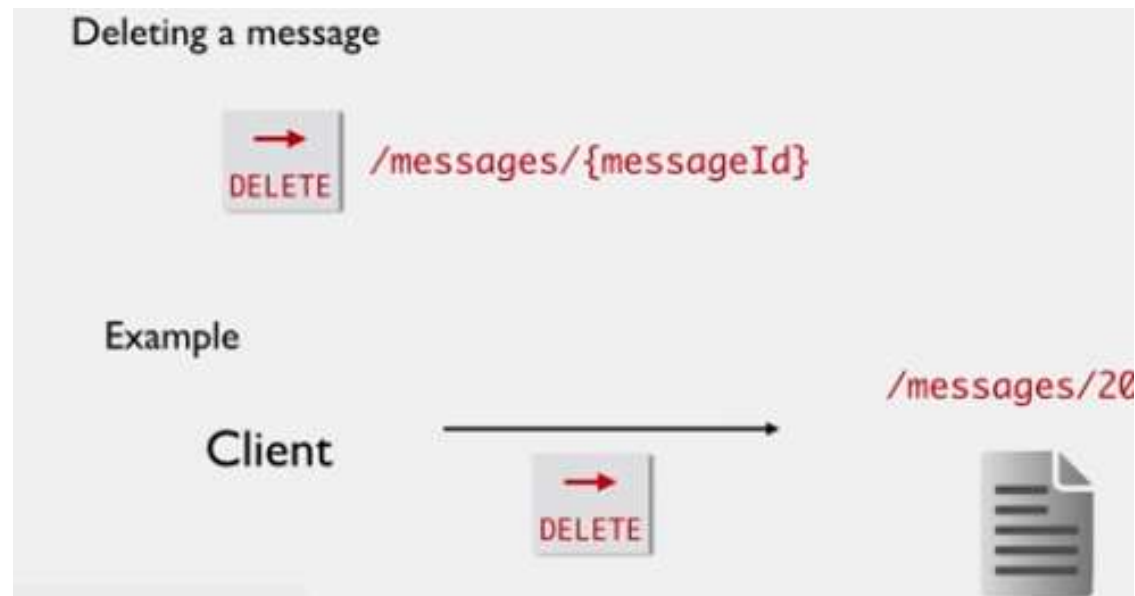
# Desarrollo Servicios REST

## Operaciones HTTP PUT



# Desarrollo Servicios REST

## Operaciones HTTP DELETE



# Desarrollo Servicios REST

## Operaciones HTTP POST

Creating a new message



/messages

Example

Client



/messages

# Desarrollo Servicios REST

## Operaciones HTTP

### Collection URI scenarios



/messages/10/comments

creates a new comment for message 10



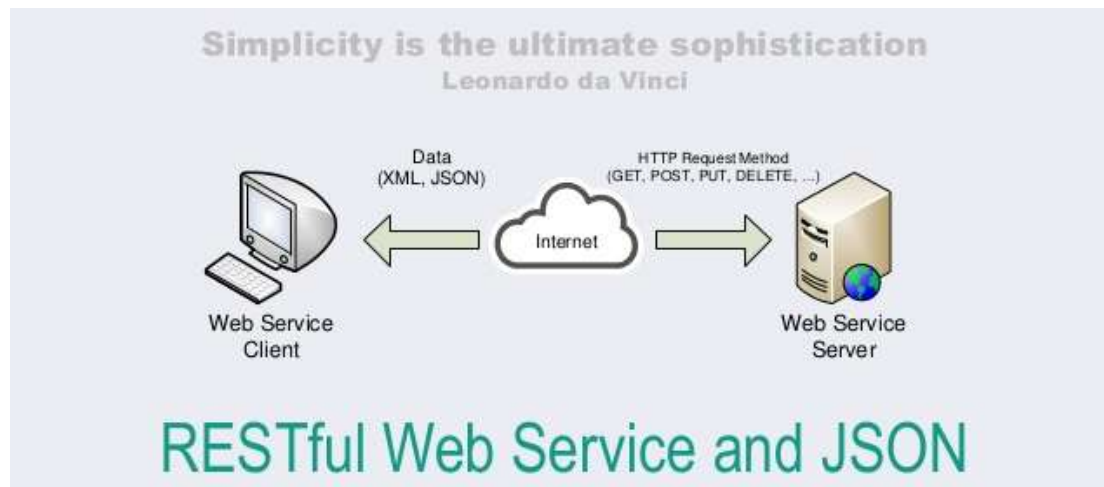
/messages/20/comments

replaces *\*all\** comments for message 20 with a new list

# Desarrollo Servicios REST

## RESTful

Arquitectura  
**REST** + Web Service = **RESTful**  
web service





# Desarrollo Servicios REST

## HATEOAS

- Hypermedia As the Engine Of Application State
- Permite incluir links a recursos en la respuesta del API, así que el cliente no necesita manejar construcción de la URI



# Desarrollo Servicios REST

## HATEOAS - the “rel” attribute

```
{
  firstname: "M.",
  lastName: "Ibrahim",
  age: 35,
  departement: "HR",
  - links: [
    - {
      rel: "self",
      href: "http://localhost:8080/person/2"
    },
    - {
      rel: "account",
      href: "http://localhost:8080/person/2/account"
    },
    - {
      rel: "profile",
      href: "http://localhost:8080/person/2/profile"
    }
  ]
}
```

# Desarrollo Servicios REST

## Richardson Maturity Model

Is this API “fully RESTful”?

Level	Características	
0	<ul style="list-style-type: none"><li>• One URI</li><li>• Message request body contains all the details</li></ul>	Is not a RESTful API
1	<ul style="list-style-type: none"><li>• Resource URI</li></ul>	Individual URIs for each resource
2	<ul style="list-style-type: none"><li>• HTTP Methods</li></ul>	Use the right HTTP methods, status codes
3	<ul style="list-style-type: none"><li>• HATEOAS</li></ul>	Responses have links that the clients can use

# Spring

Creación de implementación y cargador de la API

## Implementación de la API

- Crear proyecto
- Definir la clase principal
- Empaquetar proyecto y arrancar el servidor
- Implementar el controlador
- Implementar la entidad
- Implementar el servicio
- Implementar la capa de acceso a datos

# Creación de un servicio REST con Spring

## Requisitos de entorno

- Java 1.8
- IDE: Eclipse | IntelliJ IDEA
- Maven
- Dependencias de Spring Boot o Spring Web

# Creación de un servicio REST con Spring

## Crear el proyecto

- Crear un proyecto maven nuevo indicando en el **pom.xml** que queremos usar Spring Web con Spring Boot:

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.4.1.RELEASE</version>
</parent>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
</dependencies>
```

# Creación de un servicio REST con Spring

## Definir clase principal

- SpringApplication

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) throws Exception {
        SpringApplication.run(Application.class, args);
    }
}
```



# Creación de un servicio REST con Spring

## Empaquetar el proyecto y lanzar el servidor

- Generar ficheros .class y .jar

```
$ mvn clean package
```

- Comprobar dependencias

```
$ mvn dependency:tree
```

- Arrancar la aplicación (servidor levantado y listo para recibir peticiones)

```
$ mvn spring-boot:run
```

# Spring

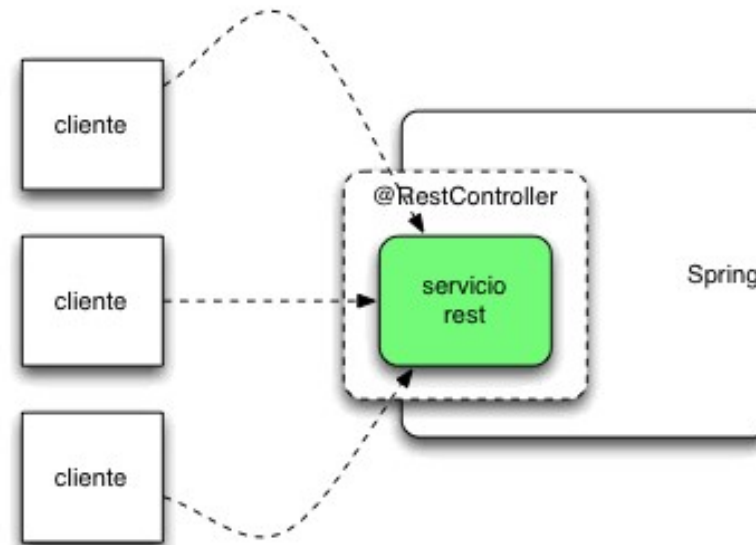
Configuración API: Entidad, Servicio, Controller

## Implementación de la API

- Crear proyecto
- Definir la clase principal
- Empaquetar proyecto y arrancar el servidor
- Implementar el controlador
- Implementar el servicio
- Implementar la entidad
- Implementar la capa de acceso a datos

## El Controlador - @RestController

- La clase anotada con `@RestController` será la encargada de gestionar las peticiones que se hagan a nuestra API.
- Indica que los datos devueltos por cada método se escribirán directamente en el cuerpo de la respuesta (response body).
- Sustituye al uso de `@Controller` + `@ResponseBody`.



## El Servicio - **@Service**

- Funcionamiento parecido a **@Controller**
- Permite que Spring reconozca a la clase anotada como servicio al escanear los componentes de la aplicación

# Implementación de la capa de Acceso a Datos

- Spring Boot + Spring data JPA

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <artifactId>spring-data-jpa</artifactId>
  <packaging>jar</packaging>
  <name>Spring Boot Spring Data JPA</name>
  <version>1.0</version>

  <parent>... </parent>

  <dependencies>
    <!-- jpa, crud repository -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
  </dependencies>
</project>
```

## @Entity

- Anotación que define objeto para persistencia en bases de datos basadas en JPA
- Permite asociar una clase a una tabla o colección
- Otras implementaciones: Spring Data, MongoDB, Spring Data Cassandra, etc...
- Anotar clases del modelo de persistencia

# La Entidad - @Entity

- Model and JPA annotations

## @Entity

```
public class Customer {  
  
    // "customer_seq" is Oracle sequence name.  
    @Id  
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "CUST_SEQ")  
    @SequenceGenerator(sequenceName = "customer_seq", allocationSize = 1, name = "CUST_SEQ")  
    Long id;  
  
    String name;  
  
        String email;  
  
    @Column(name = "CREATED_DATE")  
    Date date;  
  
    //getters and setters, constructors  
}
```



# Configuration + Database Initialization

- Configure **Oracle** data source

```
# Oracle settings
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=password
spring.datasource.driver-class-oracle.jdbc.driver.OracleDriver
```



application.properties\*

# Configuration + Database Initialization

- Configure **MongoDB** data source

```
# Spring properties
spring:
  data:
    mongodb:
      host: localhost
      port: 27017
      uri: mongodb://localhost/test

# HTTP Server
server:
  port: 4444    # HTTP (Tomcat) port
```

application.yml\*



## Implementación de la capa de acceso a datos

- Spring Boot + Spring data JPA

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongo-java-driver</artifactId>
</dependency>
```

# Implementación de la capa de acceso a datos

- Spring Data CrudRepository

```
import com.apirest.model.Customer;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;

import java.util.Date;
import java.util.List;
import java.util.stream.Stream;

public interface CustomerRepository extends CrudRepository<Customer, Long> {

    List<Customer> findByEmail(String email);

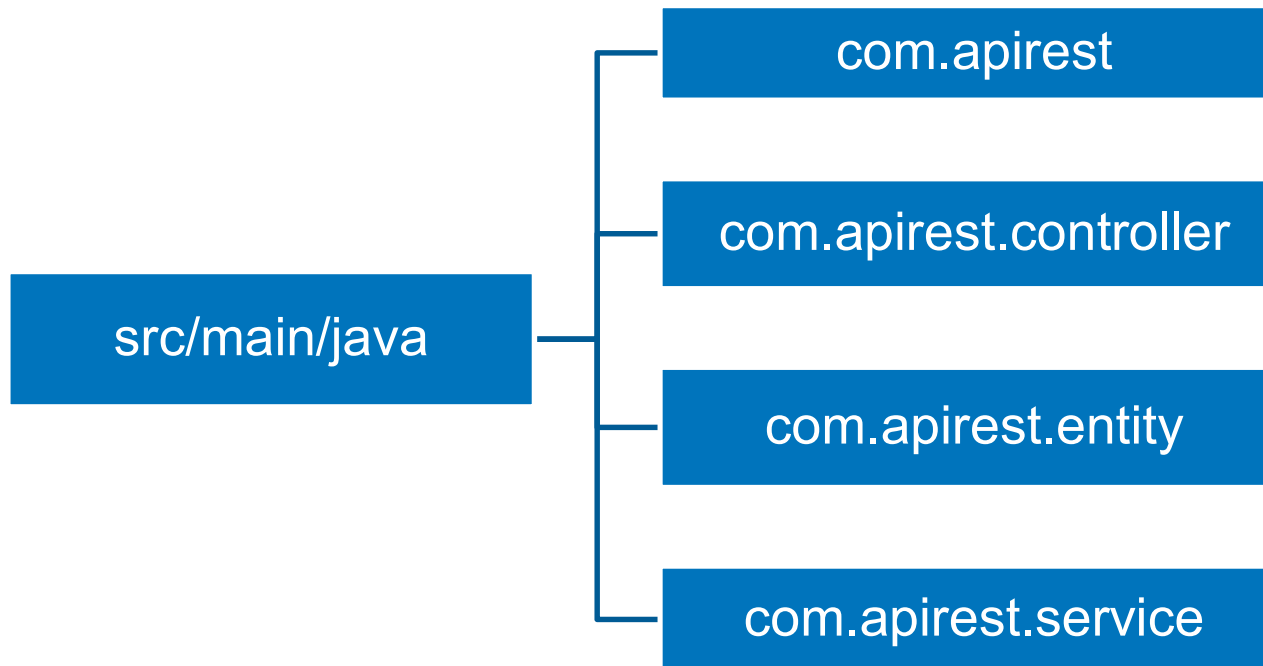
    List<Customer> findByDate(Date date);

    // custom query example and return a stream
    @Query("select c from Customer c where c.email = :email")
    Stream<Customer> findByEmailReturnStream(@Param("email") String email);

}
```

## Configuración API REST

- Estructura de la API



# Anotaciones REST API

## Anotaciones peticiones HTTP y mapeo de URLs

- **@GetMapping**
- **@PostMapping**
- **@PutMapping**
- **@DeleteMapping**
- **@PatchMapping**
- **@RequestMapping**(String URL, RequestMethod peticion)
- **@ResponseBody**

# Configuración servicio REST con Spring - Entidad

**@Entity**

```
class Employee {  
  
    private @Id @GeneratedValue Long id;  
    private String name;  
    private String role;  
  
    Employee() {}  
  
    Employee(String name, String role) {  
        this.name = name;  
        this.role = role;  
    }  
}
```

# Configuración servicio REST con Spring - Controlador

**@RestController**

```
class EmployeeController {
```

```
    private final EmployeeRepository repository;
```

```
    EmployeeController(EmployeeRepository repository) {  
        this.repository = repository;  
    }
```

**@GetMapping("/employees")**

```
List<Employee> all() {  
    return repository.findAll();  
}
```



## Configuración servicio REST con Spring – Controlador (II)

```
@PostMapping("/employees")
```

```
Employee newEmployee(@RequestBody Employee newEmployee) {  
    return repository.save(newEmployee);  
}
```

```
@GetMapping("/employees/{id}")
```

```
Employee one(@PathVariable Long id) {  
    return repository.findById(id)  
        .orElseThrow(() -> new EmployeeNotFoundException(id));  
}
```

# @RequestMapping y @ResponseBody

```
@Controller
public class SampleController {

    @Autowired
    private SampleService sampleService;

    public SampleController(SampleService sampleService) {
        this.sampleService = sampleService;
    }

    @RequestMapping(value = "/welcome/{userName}", method =
RequestMethod.GET)
    @ResponseBody
    public String welcome(
        @PathVariable("userName") String userName
    )
    {
        return sampleService.welcome(userName);
    }
}
```

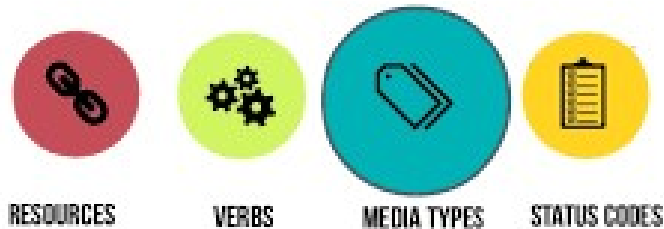
# REST Web Services

API RESTful: Mejores Prácticas

## REST-based practices

- Basado en **estándares web** y protocolo **HTTP**
- REST **no es un estándar ni un stack de tecnologías**
- Todo es un **Recurso**
- Acceso a recursos mediante peticiones HTTP (get, post, put, delete)
- Recursos identificados mediante un ID global (**URIs**)
- Diferentes **representaciones** de los recursos (XML,JSON, text...)

### REST Style consists of ...



## Buenas prácticas en el diseño de APIs

- No usar verbos en la URI sino **nombres** que identifiquen a los recursos: Una API REST representa Entidades y no acciones.
- Consistencia en la **nomencultura** de acceso a los recursos: singular o plural pero siempre el **mismo criterio**.
- Incluir la **versión** de la API en la URL.

GET/api/v1/recurso/23 HTTP/1.1

Host: our.comain.com

Accept: application/json

Action based URI

/postMessages.do?id=10



/messages/10

Resource based URI

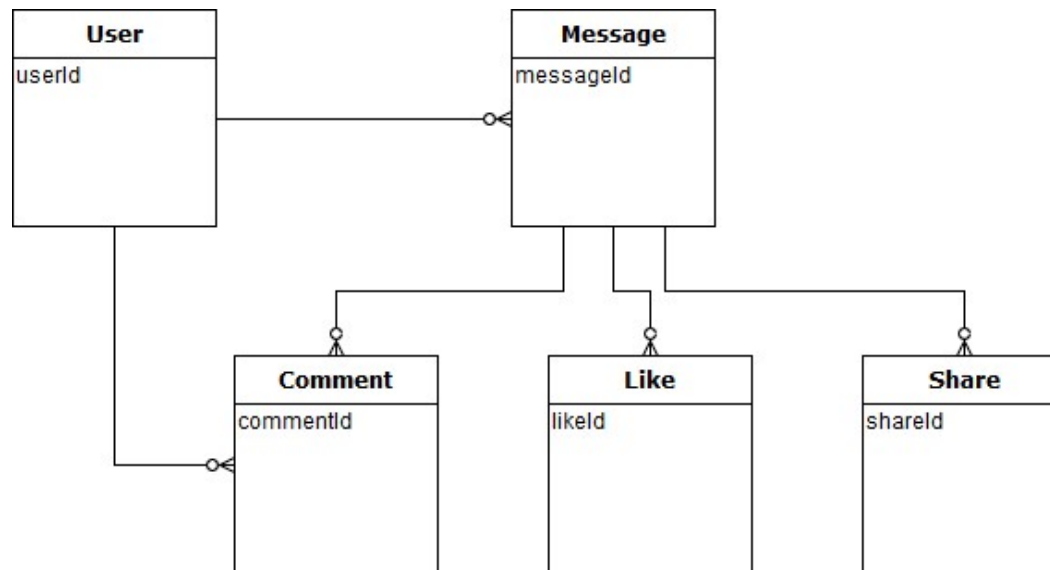
## Buenas prácticas en el diseño de APIs

- Usar **cabeceras HTTP** para especificar y validar el **formato** de la información (http-header, Content-type,...)
- Permitir que el consumidor de la API pueda elegir **formato de respuesta** (XML, **JSON**, ...)
- Utilizar los **códigos de estado** HTTP para definir errores.
- Las operaciones de una API REST no deben tener estado (deben ser *idempotentes*)
- Resource based URIs
- HTTP operations
  - ✓ GET
  - ✓ POST
  - ✓ PUT
  - ✓ DELETE
- HTTP status codes
  - ✓ 200 – Success (GET)
  - ✓ 201 – Created (POST)
  - ✓ 500 – Server error
  - ✓ 404 – Not found (GET, PUT, DELETE)
- Message headers
  - ✓ Content types: text/xml, application/json

## Buenas prácticas en el diseño de APIs

- Representar **relación** entre dos recursos (1:N) como **subrecurso**
- Una entidad en una API no tiene por qué representar una tabla de nuestro modelo entidad-relación

`/profiles/{profileName}/comments/`  
`/messages/{messageId}/likes/{likeId}`



## Buenas prácticas en el diseño de APIs

- Incluir funcionalidades de **filtrado**, **ordenado** y **acceso a la información** de cada campo.

### Query Parameters

/messages/?offset=5&limit=10



starting point



page size

### Custom Filters

/messages/?year=2020



Filtering Collection



## Buenas prácticas en el diseño de APIs

```
{
  firstname: "M.",
  lastName: "Ibrahim",
  age: 35,
  departement: "HR",
  - links: [
    - {
      rel: "self",
      href: "http://localhost:8080/person/2"
    },
    - {
      rel: "account",
      href: "http://localhost:8080/person/2/account"
    },
    - {
      rel: "profile",
      href: "http://localhost:8080/person/2/profile"
    }
  ]
}
```

### HATEOAS Hypermedia As the Engine Of Application State

- permite **incluir links** a recursos en la respuesta del API
- parte de la información devuelta serán identificadores únicos en forma de hipervínculos a otros recursos asociados

# Buenas prácticas en el diseño de APIs

## Relativas a la Seguridad y Privacidad de los datos:

- No proporcionar información sensible (Hashcode de una password,...)
- No utilizar primary keys o foreign keys como parámetros
- Usar tokens de seguridad como JWT (JSON Web Token) para asegurar la identidad en ambos extremos ([www.jwt.io/](http://www.jwt.io/))
- Exigir credenciales de seguridad únicamente para aquellas operaciones que produzcan modificaciones como: creación de recursos, borrado o modificación de un registro.

# Buenas prácticas en el diseño de APIs

- Utilizar Herramientas para la creación, diseño y documentación de APIs

user

Show/Hide

List Operations

Expand Operations

Raw

word

Show/Hide

List Operations

Expand Operations

Raw

GET

/word.json/{word}/entries

Return entries for a word

GET

/word.json/{word}/examples

Returns examples for a word

POST

/word.json/{word}/examples

Fetches examples for a word

POST

/word.json/{word}/wordForms

Adds a Relationship Map to a word

GET

/word.json/{word}/wordForms

Returns other forms of a word

DELETE

/word.json/{word}/wordForms

Deletes a relationship from a word

GET

/word.json/{word}

Given a word as a string, returns the WordObject that represents it

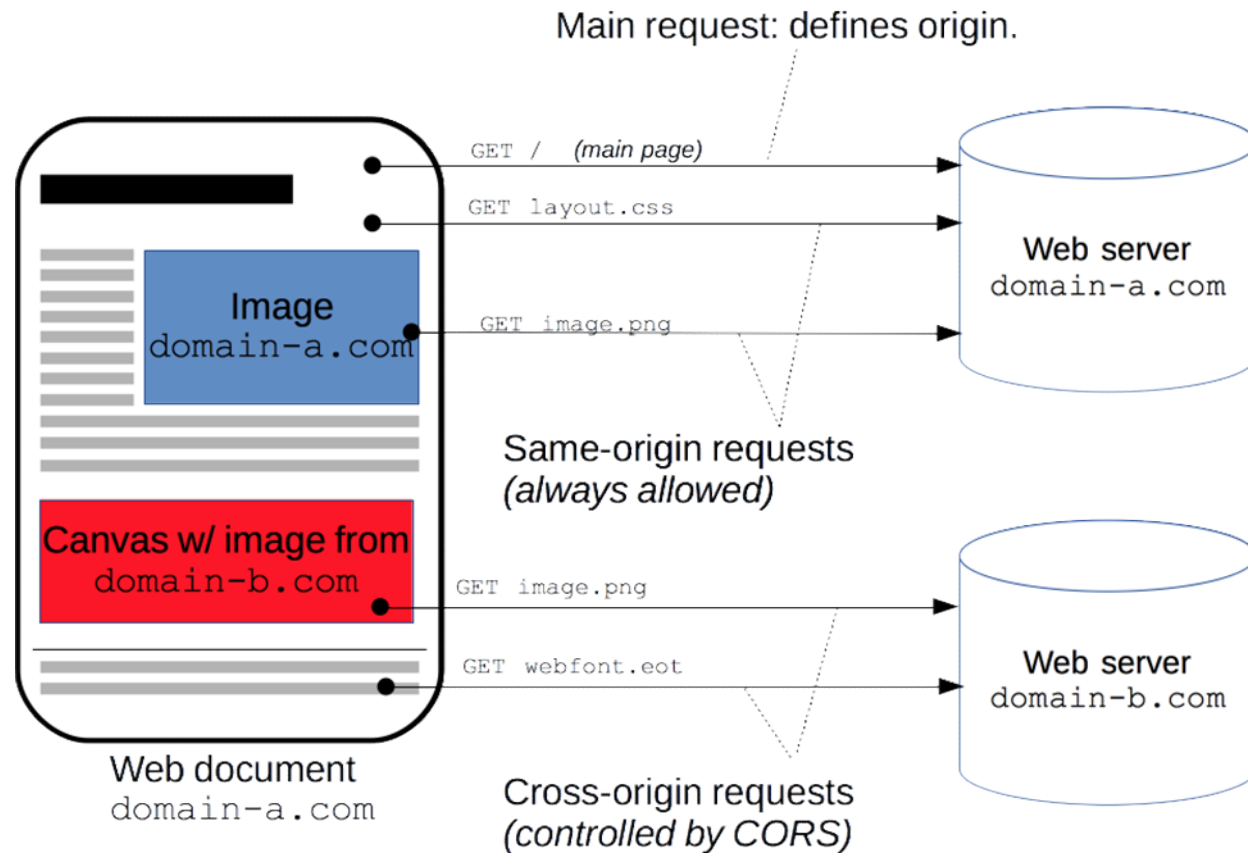
Parameters

Parameter	Value	Description
word	<input type="text" value="(required)"/>	String value of WordObject to return
useCanonical	<input type="text"/>	If true will try to return the correct word root ('cats' -> 'cat'). If false returns exactly what was requested.
includeSuggestions	<input type="text"/>	Return suggestions (for correct spelling, case variants, etc.)
shouldCreate	<input type="text"/>	Create word if not existing

## CORS | Cross-Origin Resource Sharing

- El **modelo de seguridad** de las aplicaciones web **no permite** en principio realizar **peticiones** asíncronas **entre dominios**
- Problema: una página de `http://localhost` no puede hacer una petición AJAX a Google
- Solución: CORS añade funcionalidades nuevas a las peticiones AJAX como las **peticiones entre dominios (cross-site)**

# CORS | Cross-Origin Resource Sharing



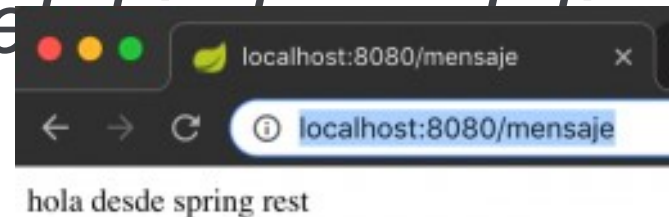
## CORS | Cross-Origin Resource Sharing



## Servicio REST con Spring Boot y @RestController

- **import**  
`org.springframework.web.bind.annotation.GetMapping;`
- **import**  
`org.springframework.web.bind.annotation.RestController;`

Creando el controlador nos será suficiente con arrancar la aplicación de Spring Boot y solicitar la URL.



• `// Clase anotada con RestController`

## Servicio REST con Spring Framework

### CORS Policy

```
<html>
<head> <script type="text/javascript" src="jquery-
3.3.1.min.js"></script> <script type="text/javascript">
  $(document).ready(function() {
$.get("http://localhost:8080/mensaje",function(datos)
  { console.log(datos); }) })
</script> </head>
<body></body>
</html>
```

Si cargamos esta página veremos un error en la consola que nos restringe el acceso debido a que estamos realizando una petición AJAX desde JavaScript y estas peticiones por defecto están limitadas a ficheros JavaScript que nos descarguemos desde el mismo servidor.

✖ Access to XMLHttpRequest at 'http://localhost:8080/mensaje' from 1.html:1 origin 'null' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.



## Servicio REST con Spring Framework

### Anotación `@CrossOrigin`

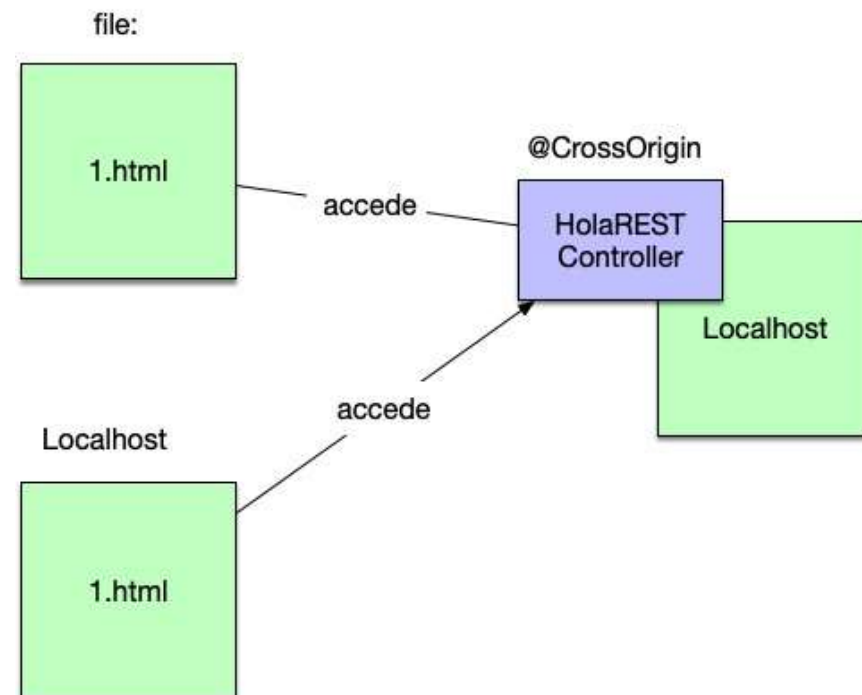
Para solventar CORS policy es suficiente con modificar el servicio de Spring y añadir una cabecera `@CrossOrigin` para que nos permita el acceso desde otras ubicaciones.

- **import**  
**`org.springframework.web.bind.annotation`**  
**`.CrossOrigin;`**
- **import**  
*`org.springframework.web.bind.annotation.GetMapping;`*
- **import**  
*`org.springframework.web.bind.annotation.RequestMapping;`*

## CORS | Cross-Origin Resource Sharing

### Spring REST CORS


- Siempre que tengamos recursos REST y queramos acceder a ellos debemos usar Spring REST CORS y abrir el acceso remoto ya que por defecto los datos de nuestros servicios no estarán accesibles.



## Spring REST Client con **RestTemplates**

- `import org.springframework.web.client.RestTemplate;`
- 
- `@SpringBootApplication`
- `public class ClienteApplication implements CommandLineRunner {`
- 
- `public static void main(String[] args) {`
- 
- `SpringApplication app = new SpringApplication(ClienteApplication.class);`
- `app.setWebEnvironment(false);`
- `app.run(args);}`
- 
- `@Override`
- `public void run(String... args) throws Exception {`
- `RestTemplate plantilla = new RestTemplate();`
- `String resultado = plantilla.getForObject("http://localhost:8080/hola", String.class);`
- `System.out.println(resultado);`

Creamos un objeto de tipo **RestTemplate** para invocar de forma directa una URL y acceder a los datos que se encuentran en /hola



# REST API

Documentación de Microservicios

## Beneficios de Documentar APIs

- Evitar problemas por falta de planificación y diseño previo del componente / API
- Evitar inconsistencia entre los objetos y métodos
- Evitar agujeros de seguridad
- Uso de herramientas que tienen en cuenta la usabilidad y necesidades de los consumidores/aplicaciones que van a utilizar los servicios
- Realizar mocks testeables
- Posibilitar el versionado
- Crear de forma conjunta al desarrollo de la documentación
- Herramientas para la creación y diseño de APIs
- **RAML y Swagger**

## Herramientas para la creación y diseño de APIs

### **RAML (RESTful API Modeling Language)**

- Lenguaje de definición de APIs que permite escribir su especificación siguiendo un estándar
- Lenguaje de modelado para definir APIs REST de sintaxis sencilla y fácilmente comprensibles para seres humanos y software
- Especificación no propietaria e independiente basada en YAML y JSON
- Permite definir versión, recursos, métodos, parámetros de URL, seguridad respuestas, tipos de medios y otros componentes HTTP básicos
- Genera la documentación de la API, casos de prueba, implementa un mock para acelerar el desarrollo y genera el esqueleto de nuestra aplicación
- Permite definir las respuestas y ejemplos escritos en la especificación como documentación

For every API, start by defining which version of RAML you are using, and then document basic characteristics of your API - the title, baseURI, and version.

Create and pull in namespaced, reusable libraries containing data types, traits, resource types, schemas, examples, & more.

Annotations let you add vendor specific functionality without compromising your spec

Traits and resourceTypes let you take advantage of code reuse and design patterns

Easily define resources and methods, then add as much detail as you want. Apply traits and other patterns, or add parameters and other details specific to each call.

Describe expected responses for multiple media types and specify data types or call in pre-defined schemas and examples. Schemas and examples can be defined via a data type, in-line, or externalized with !include.

Write human-readable, markdown-formatted descriptions throughout your RAML spec, or include entire markdown documentation sections at the root.

```
1  #%RAML 1.0
2  title: World Music API
3  baseUri: http://example.api.com/{version}
4  version: v1
5
6  uses:
7    Songs: libraries/songs.raml
8
9  annotationTypes:
10   monitoringInterval:
11     parameters:
12       value: integer
13
14  traits:
15   secured: !include secured/accessToken.raml
16
17  /songs:
18   is: [ secured ]
19   get:
20     (monitoringInterval): 30
21     queryParameters:
22       genre:
23         description: filter the songs by genre
24   post:
25   /{songId}:
26     get:
27       responses:
28         200:
29           body:
30             application/json:
31               type: Songs.Song
32             application/xml:
33               schema: !include schemas/songs.xml
34               example: !include examples/songs.xml
```

#### Songs Library

```
1  #%RAML 1.0 Library
2  types:
3    Song:
4      properties:
5        title: string
6        length: number
7    Album:
8      properties:
9        title: string
10       songs: Song[]
11    Musician:
12      properties:
13        name: string
14       discography: (Song | Album)[]
```

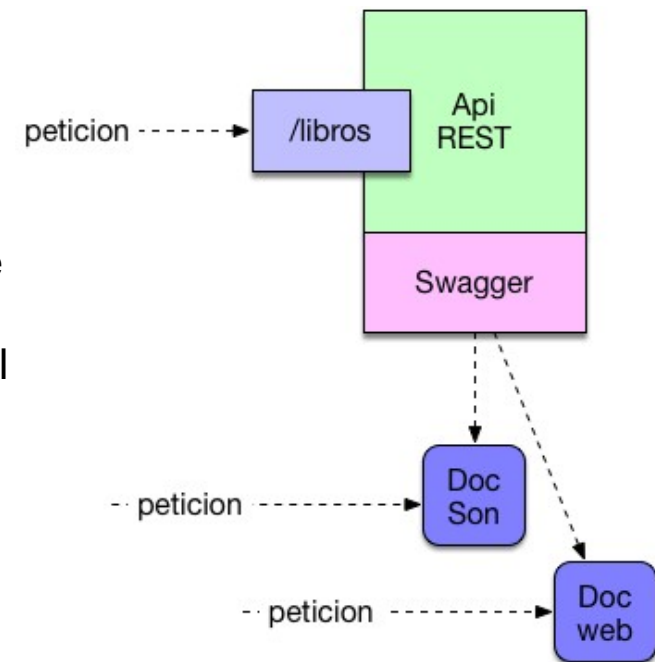
#### songs.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3    elementFormDefault="qualified" attributeFormDefault="unqualified">
4    <xs:element name="song">
5      <xs:complexType>
6        <xs:sequence>
7          <xs:element name="title" type="xs:string"/>
8          </xs:element>
9          <xs:element name="artist" type="xs:string"/>
10         </xs:element>
11        </xs:sequence>
12      </xs:complexType>
13    </xs:element>
14  </xs:schema>
```

## Herramientas para la creación y diseño de APIs

### Swagger

- Permite describir, producir, consumir y visualizar APIs RESTful
- Framework open source para **generar documentación** de APIs RESTful
- Interfaz visual a modo de sandbox para testear llamadas al API, y consulta de su documentación en el navegador
- Código del servidor sincronizado automáticamente con la documentación generada
- Generación de documentación para Java, Javascript, Ruby, PHP, Scala, ActionScript y su sandbox correspondiente





# Swagger

The screenshot displays the Swagger API Explorer interface for an API named 'my-awesome-api.com'. The 'word' API is selected, showing a list of endpoints with their HTTP methods and descriptions. Below the endpoints, the 'Parameters' section is expanded, showing a table with columns for 'Parameter', 'Value', and 'Description'. The 'word' parameter is required and is a string. Other parameters include 'useCanonical', 'includeSuggestions', and 'shouldCreate'. A 'Try it out!' button is visible below the parameters table. The bottom of the interface shows a list of endpoints for the 'word' API, including definitions, stats, top example, contextual lookup, comment count, related words, listed in, and listed in count.

**word** Show/Hide List Operations Expand Operations Raw

- GET /word.json/{word}/entries Return entries for a word
- GET /word.json/{word}/examples Returns examples for a word
- POST /word.json/{word}/examples Fetches examples for a word
- POST /word.json/{word}/wordForms Adds a Relationship Map to a word
- GET /word.json/{word}/wordForms Returns other forms of a word
- DELETE /word.json/{word}/wordForms Deletes a relationship from a word
- GET /word.json/{word} Given a word as a string, returns the WordObject that represents it

**Parameters**

Parameter	Value	Description
word	(required)	String value of WordObject to return
useCanonical		If true will try to return the correct word root ('cats' -> 'cat'). If false returns exactly what was requested.
includeSuggestions		Return suggestions (for correct spelling, case variants, etc.)
shouldCreate		Create word if not existing

Try it out!

- GET /word.json/{word}/definitions Return definitions for a word
- GET /word.json/{word}/stats Returns word statistics
- GET /word.json/{word}/topExample Returns a top example for a word
- GET /word.json/{word}/contextualLookup Returns definitions for a word based on the sentence in which it is found
- POST /word.json/{word}/contextualLookup Returns definitions for a word based on the sentence in which it is found
- GET /word.json/{word}/commentCount Returns the number of comments on a word
- GET /word.json/{word}/related Return related words (thesaurus data) for a word
- GET /word.json/{word}/listedIn Returns WordLists containing a word
- GET /word.json/{word}/listedInCount Returns a count of lists a word appears in

## Herramientas para la creación y diseño de APIs

### Swagger

1. **Dependencias** del archivo de configuración (pom.xml) para habilitar Swagger2 en una aplicación Spring Boot:

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.7.0</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.7.0</version>
</dependency>
```

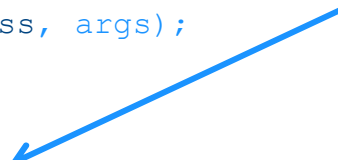
## Herramientas para la creación y diseño de APIs

### Swagger

#### 2. Habilitar Swagger2 en la aplicación Spring Boot principal (main) con la anotación `@EnableSwagger2`:

```
import springfox.documentation.swagger2.annotations.EnableSwagger2;
@SpringBootApplication
@EnableSwagger2
public class SwaggerDemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(SwaggerDemoApplication.class, args);
    }
    @Bean
    public Docket productApi() {
        return
            new Docket(DocumentationType.SWAGGER_2).select()
                .apis(RequestHandlerSelectors.basePackage("com.tutorialspoint.swaggerdemo"))
                .build();
    }
}
```

Creamos el bean Docket para configurar Swagger2 en la aplicación. Es necesario definir el base package para configurar APIs REST para Swagger2.



## Swagger2

### Documentación formato Web

<http://localhost:8080/swagger-ui.html>



### Api Documentation

Api Documentation

[Apache 2.0](#)

**controlador-persona : Controlador Persona**

[ BASE URL: / , API VERSION: 1.0 ]

#### controlador-persona : Controlador Persona

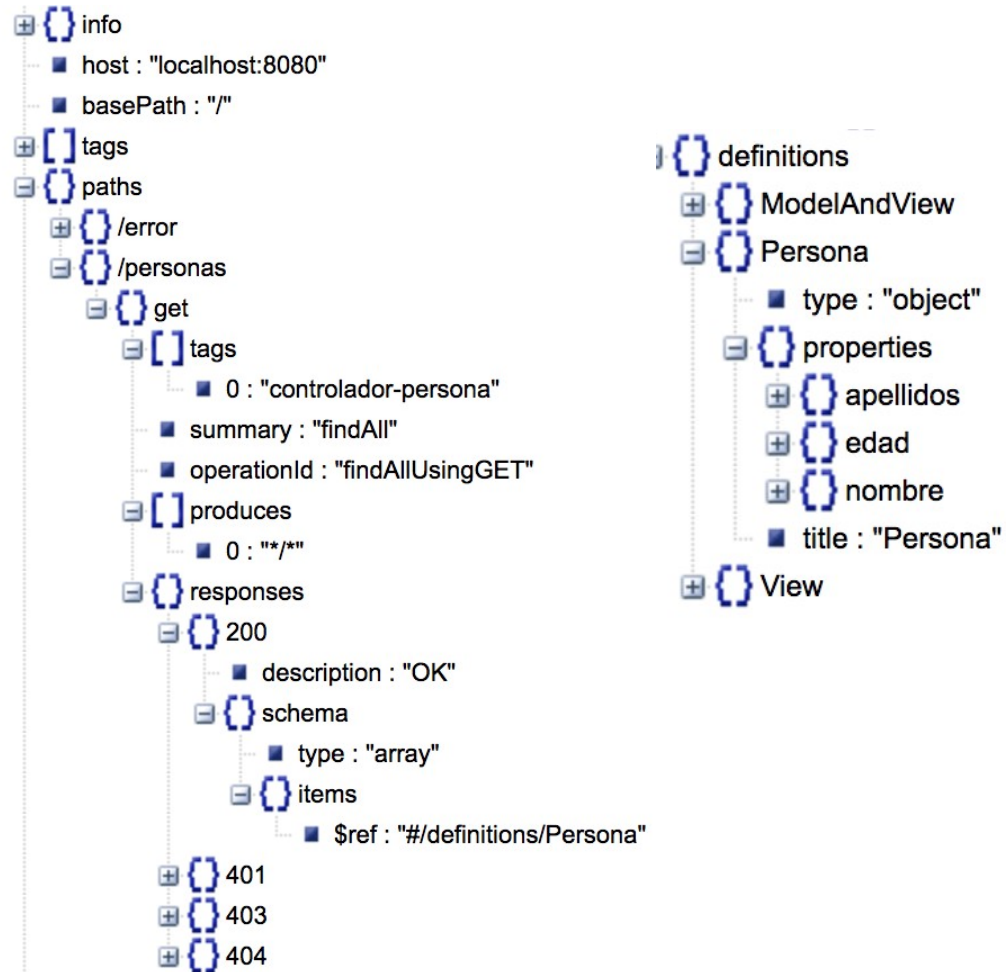
GET	/personas
Response Class (Status 200)	
OK	
Model	Example Value
<pre>[   {     "apellidos": "string",     "edad": 0,     "nombre": "string"   } ]</pre>	

## swagger.json

## Documentación formato JSON /v2/api-docs

```
{
  "swagger": "2.0",
  "info": {
    "description": "Api Documentation",
    "version": "1.0",
    "title": "Api Documentation",
    "termsOfService": "urn:tos",
    "contact": {}
  },
  "license": {
    "name": "Apache 2.0",
    "url": "http://www.apache.org/licenses/LICENSE-2.0"
  },
  "host": "localhost:8080",
  "basePath": "/",
  "tags": [
    {
      "name": "basic-error-controller",
      "description": "Basic Error Controller"
    },
    {
      "name": "controlador-persona",
      "description": "Controlador Persona"
    }
  ],
  "paths": {
    "/error": {
      "get": {
        "tags": [
          "basic-error-controller"
        ],
        "summary": "error",
        "operationId": "errorUsingGET",
        "produces": [
          "*"
        ],
        "responses": {
          "200": {
            "description": "OK",
            "schema": {
              "type": "object",
              "additionalProperties": {
                "type": "object"
              }
            }
          },
          "401": {
            "description": "Unauthorized"
          },
          "403": {
            "description": "Forbidden"
          },
          "404": {
            "description": "Not Found"
          }
        },
        "head": {
          "tags": [
            "basic-error-controller"
          ],
          "summary": "error",
          "operationId": "errorUsingHEAD",
          "consumes": [
            "application/json"
          ],
          "produces": [
            "*"
          ],
          "responses": {
            "200": {
              "description": "OK",
              "schema": {
                "type": "object",
                "additionalProperties": {
                  "type": "object"
                }
              }
            },
            "204": {
              "description": "No Content"
            },
            "401": {
              "description": "Unauthorized"
            },
            "403": {
              "description": "Forbidden"
            }
          },
          "post": {
            "tags": [
              "basic-error-controller"
            ],
            "summary": "error",
            "operationId": "errorUsingPOST",
            "consumes": [
              "application/json"
            ],
            "produces": [
              "*"
            ],
            "responses": {
              "200": {
                "description": "OK",
                "schema": {
                  "type": "object",
                  "additionalProperties": {
                    "type": "object"
                  }
                }
              },
              "201": {
                "description": "Created"
              },
              "401": {
                "description": "Unauthorized"
              },
              "403": {
                "description": "Forbidden"
              },
              "404": {
                "description": "Not Found"
              }
            },
            "put": {
              "tags": [
                "basic-error-controller"
              ],
              "summary": "error",
              "operationId": "errorUsingPUT",
              "consumes": [
                "application/json"
              ],
              "produces": [
                "*"
              ],
              "responses": {
                "200": {
                  "description": "OK",
                  "schema": {
                    "type": "object",
                    "additionalProperties": {
                      "type": "object"
                    }
                  }
                },
                "201": {
                  "description": "Created"
                },
                "401": {
                  "description": "Unauthorized"
                },
                "403": {
                  "description": "Forbidden"
                },
                "404": {
                  "description": "Not Found"
                }
              },
              "delete": {
                "tags": [
                  "basic-error-controller"
                ],
                "summary": "error",
                "operationId": "errorUsingDELETE",
                "produces": [
                  "*"
                ],
                "responses": {
                  "200": {
                    "description": "OK",
                    "schema": {
                      "type": "object",
                      "additionalProperties": {
                        "type": "object"
                      }
                    }
                  },
                  "204": {
                    "description": "No Content"
                  },
                  "401": {
                    "description": "Unauthorized"
                  },
                  "403": {
                    "description": "Forbidden"
                  }
                },
                "patch": {
                  "tags": [
                    "basic-error-controller"
                  ],
                  "summary": "error",
                  "operationId": "errorUsingPATCH",
                  "consumes": [
                    "application/json"
                  ],
                  "produces": [
                    "*"
                  ],
                  "responses": {
                    "200": {
                      "description": "OK",
                      "schema": {
                        "type": "object",
                        "additionalProperties": {
                          "type": "object"
                        }
                      }
                    },
                    "204": {
                      "description": "No Content"
                    },
                    "401": {
                      "description": "Unauthorized"
                    },
                    "403": {
                      "description": "Forbidden"
                    }
                  }
                },
                "options": {
                  "tags": [
                    "basic-error-controller"
                  ],
                  "summary": "error",
                  "operationId": "errorUsingOPTIONS",
                  "consumes": [
                    "application/json"
                  ],
                  "produces": [
                    "*"
                  ],
                  "responses": {
                    "200": {
                      "description": "OK",
                      "schema": {
                        "type": "object",
                        "additionalProperties": {
                          "type": "object"
                        }
                      }
                    },
                    "204": {
                      "description": "No Content"
                    },
                    "401": {
                      "description": "Unauthorized"
                    },
                    "403": {
                      "description": "Forbidden"
                    }
                  }
                }
              }
            }
          },
          "personas": {
            "get": {
              "tags": [
                "controlador-persona"
              ],
              "summary": "findAll",
              "operationId": "findAllUsingGET",
              "produces": [
                "*"
              ],
              "responses": {
                "200": {
                  "description": "OK",
                  "schema": {
                    "type": "array",
                    "items": {
                      "$ref": "#/definitions/Persona"
                    }
                  }
                },
                "401": {
                  "description": "Unauthorized"
                },
                "403": {
                  "description": "Forbidden"
                },
                "404": {
                  "description": "Not Found"
                }
              }
            }
          },
          "definitions": {
            "ModelAndView": {
              "type": "object",
              "properties": {
                "empty": {
                  "type": "boolean"
                },
                "model": {
                  "type": "object",
                  "modelMap": {
                    "type": "object",
                    "additionalProperties": {
                      "type": "object"
                    },
                    "reference": {
                      "type": "boolean"
                    },
                    "status": {
                      "type": "string",
                      "enum": [
                        "100",
                        "101",
                        "102",
                        "103",
                        "200",
                        "201",
                        "202",
                        "203",
                        "204",
                        "205",
                        "206",
                        "207",
                        "208",
                        "226",
                        "300",
                        "301",
                        "302",
                        "303",
                        "304",
                        "305",
                        "307",
                        "308",
                        "400",
                        "401",
                        "402",
                        "403",
                        "404",
                        "405",
                        "406",
                        "407",
                        "408",
                        "409",
                        "410",
                        "411",
                        "412",
                        "413",
                        "414",
                        "415",
                        "416",
                        "417",
                        "418",
                        "419",
                        "420",
                        "421",
                        "422",
                        "423",
                        "424",
                        "426",
                        "428",
                        "429",
                        "431",
                        "451",
                        "500",
                        "501",
                        "502",
                        "503",
                        "504",
                        "505",
                        "506",
                        "507",
                        "508",
                        "509",
                        "510",
                        "511"
                      ]
                    },
                    "view": {
                      "$ref": "#/definitions/View",
                      "viewName": {
                        "type": "string"
                      },
                      "title": "ModelAndView",
                      "Persona": {
                        "type": "object",
                        "properties": {
                          "apellidos": {
                            "type": "string"
                          },
                          "edad": {
                            "type": "integer",
                            "format": "int32"
                          },
                          "nombre": {
                            "type": "string"
                          }
                        },
                        "title": "Persona",
                        "View": {
                          "type": "object",
                          "properties": {
                            "contentType": {
                              "type": "string"
                            },
                            "title": "View"
                          }
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

## Swagger2 Documentación con visor JSON [/v2/api-docs](#)



# Recursos

<https://spring.io/guides/gs/spring-boot/>

<https://spring.io/docs>